Programming Assignment Report - #4

Yonwoo Choi (2022-28614)
yhugestar@snu.ac.kr
Seoul National University - Advanced Graphics (Spring 2023)

Abstract

This programming assignment focuses on the development of a Deep Reinforcement Learning (DeepRL) framework for training control policies in physically simulated environments. The main goal is to design an efficient controller that enables a Half Cheetah model to navigate through environments, and to understand the effect of different hyperparameters on the learning process.

In Step 1, the assignment involves experimenting with the training of controllers by modifying critical hyperparameters. Particularly, the effects of varying the discount factor (gamma) and exploration noise are investigated. Three experiments are conducted to observe the outcomes of employing a larger, smaller, and default discount factor. Similarly, another set of experiments is executed to examine the influence of different levels of exploration noise. For all the experiments, the performance metrics and behavior of the Half Cheetah are analyzed, and the visual demonstrations of the experiments are submitted in the form of video clips.

In Step 2, the assignment takes a step further by introducing a more challenging environment with obstacles on the ground. This new environment tests the agility and navigation capabilities of the Half Cheetah. The default setup is anticipated to struggle in efficiently learning a successful controller for this modified environment. As a result, the assignment requires altering specific settings, such as the discount factor, exploration noise, observation (including additional observations like distance to obstacles), and the reward function. The objective remains to train a controller that enables the Half Cheetah to adeptly navigate through the obstacle-laden environment. The performance is measured by the average distance that the Half Cheetah can reach within 10 seconds, and a video showcasing the most successful result is recorded and submitted.

1 Learning controllers with various hyper-parameters

1.1 Altering the discount factor

Since step 1 is tuning hyperparameters in an environment without obstacles, the given skeleton code was ran using the command.

```
python main.py --model_path default.zip
```

Three experiments were conducted to observe the effects of altering the discount factor (gamma). **Listing 1** shows the line that was modified to test the effects of gamma.[1]

Listing 1: Discount Factor

```
model = PPO("MlpPolicy", env, verbose=1, policy_kwargs=↩
    policy_kwargs, tensorboard_log="./logs/", gamma=0.99)
```

The total number of time steps was kept to the default value of 500000 steps, and the exploration noise was given the default value of 1. The three experiments were:

- 1.Discount Factor: 1.00 - **larger** discount factor than the default setup [2]
- 2.Discount Factor: 0.10 - **smaller** discount factor than the default setup
- 3.Discount Factor: 0.99 - **default** setup

The discount factor, denoted as γ (gamma), plays a crucial role in determining how much the agent values future rewards compared to immediate ones. It is used to calculate the discounted cumulative future reward.

The value of γ ranges from 0 to 1:

- When γ is close to 0, the agent is myopic and primarily focused on immediate rewards, neglecting the future.
- When γ is close to 1, the agent considers future rewards nearly as much as immediate rewards, effectively trying to maximize its long-term gains.
- When γ is exactly 1, the agent values future rewards equally to the immediate ones.

In these experiments, one of the most important metrics to consider is the mean episode reward (`ep_rew_mean`). This metric represents the average total reward received by the agent per episode, and it is a direct measure of the agent's performance. Additionally, the `explained_variance` represents how much variance of the returns is captured by the value function.

- **Experiment 1 ($\gamma = 1.00$):** The mean episode reward is 17.86, which suggests that the agent performs reasonably. The `explained_variance` is 0.644, indicating that a significant portion of the variance in returns is captured by the value function.
- **Experiment 2 ($\gamma = 0.1$):** The mean episode reward is -26.23, which is significantly lower than in Experiment 1. This implies that the agent performs poorly when future rewards are heavily discounted. The `explained_variance` is very high at 0.997, but it does not translate into higher rewards, which is what ultimately matters.
- **Experiment 3 ($\gamma = 0.99$):** The mean episode reward is 138.02, which is substantially higher than in

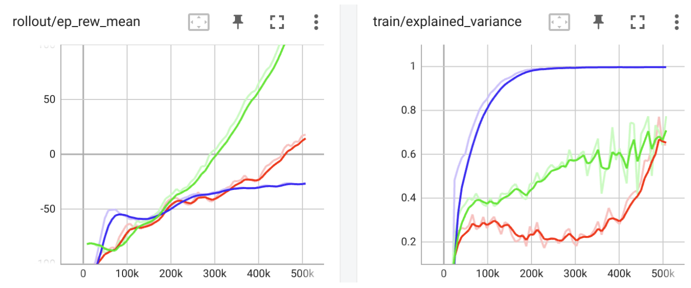

Figure 1: TensorBoard results for different discount factors (γ). The left side shows the `mean episode reward`, and the right side depicts `explained variance`. The blue line ($\gamma = 0.1$), red line ($\gamma = 1.0$), and green line ($\gamma = 0.99$) are shown. The green line achieves the highest `ep_rew_mean` and a relatively high `explained_variance`, indicating better performance.

both Experiment 1 and 2. This suggests that the default value of the discount factor allows the agent to perform much better. The `explained_variance` is 0.774, showing that the value function is capturing a good portion of the variance in returns.

From the results, it is evident that the discount factor plays a critical role in the performance of the agent. A value of $\gamma = 0.99$ appears to be the most balanced in terms of considering both immediate and future rewards, leading to the highest mean episode reward. On the other hand, setting $\gamma = 0.1$ causes the agent to be myopic and perform poorly since it heavily discounts future rewards.

1.2 Altering the exploration noise

Three experiments were conducted, excluding the default experiment, to observe the effects of altering the exploration noise. The rest of the parameters were kept at their default values, with the discount factor set to 0.99. To modify the exploration noise, the standard deviation of the policy's action distribution is adjusted through the `log_std_init` parameter in the `policy_kwargs` dictionary. The `log_std_init` parameter sets the initial log standard deviation for the policy's action distribution. This is crucial because a higher standard deviation means that the policy will take more diverse actions, exploring the environment more broadly, while a lower standard deviation leads to more focused exploration around the mean action values.[3] **Listing 2** shows the line that was modified to test the effects of exploration noise.

Listing 2: Exploration Noise

```
policy_kwargs = dict(log_std_init = 0)
```

In this example, `log_std_init` is set to 0, which corresponds to a standard deviation of 1 in the action distribution since $\exp(0) = 1$. By changing this value, one can control the exploration behavior of the agent.

- **Experiment 1 ($\varepsilon = 0.135$):** The mean episode reward is 373.242, which suggests that the agent performs relatively good. The `explained_variance` is 0.848, indicating that a significant portion of the variance in returns is captured by the value function.

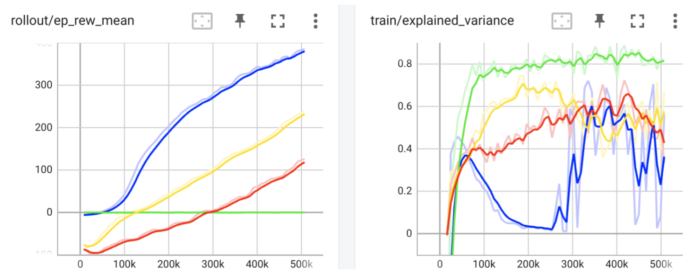

Figure 2: Tensorboard results for different exploration noise(ε), to observe the effects of altering the exploration noise. The rest of the parameters were kept at their default values, with the `exploration noise` is set to 1. The blue line ($\varepsilon = 0.135$), red line ($\varepsilon = 1.00$), and green line ($\varepsilon = 0.00673$) are shown. The blue line achieves the highest `ep_rew_mean` and a relatively high `explained_variance`, indicating better performance.

- **Experiment 2 ($\varepsilon = 1.00$):** This is the default value from the skeleton code. The mean episode reward is 125.816, which is significantly lower than in Experiment 1. This implies that the agent performs poorly when the exploration noise is higher. The `explained_variance` is very high at 0.334, but it does not translate into higher rewards, which is what ultimately matters.
- **Experiment 3 ($\varepsilon = 0.00673$):** The mean episode reward is -0.2208, which is substantially higher than in both Experiment 1 and 2. This suggests that the default value of the discount factor allows the agent to perform much better. The `explained_variance` is 0.821, showing that the value function is capturing a good portion of the variance in returns.

2 Half Cheetah with obstacles

In this task, obstacles in the Half Cheetah environment made the task more challenging. The primary objective was to train a controller that enables our cheetah to run while avoiding obstacles. Various settings were modified to tackle this scenario, namely the discount factor, exploration noise, observation, and the reward function.

2.1 Discount factor

In our experiments, I found that using a discount factor of 1.00 resulted in better long-term performance compared to the default value of 0.99. A higher discount factor allowed the agent to place more value on future rewards, which encouraged the agent to find a balance between avoiding obstacles and moving forward efficiently.

2.2 Exploration noise

I've experimented with different levels of exploration noise to observe its impact on the performance of Half Cheetah. I found that using an exploration noise of 0.2 was optimal for this task. Lower noise levels resulted in the agent not exploring enough, and hence, not learning to avoid obstacles effectively. On the other hand, higher noise levels led to erratic behavior.

2.3 Observation

The default observations were modified to include the distance and height of the nearest obstacle. This extra information helped the agent to plan its jumps over obstacles more effectively, and I observed that the agent was able to avoid obstacles with higher consistency after including these observations.

2.4 Reward

I have defined a custom reward function that considers not only the forward movement but also the upright posture of the cheetah, the proximity to obstacles, and rewards for jumping over obstacles. The reward function is as follows:

```
def step(self, action):
    x_position_before = self.data.qpos[0]
    self.do_simulation(action, self.frame_skip)
    x_position_after = self.data.qpos[0]
    z_position_after = self.data.qpos[1]
    x_velocity = (x_position_after - x_position_before) / self.dt
    ctrl_cost = self.control_cost(action)
    forward_reward = self._forward_reward_weight * x_velocity
    observation = self._get_obs()

    # Custom reward function
    rooty_angle = self.data.qpos[2]
    angle_penalty = -abs(rooty_angle)

    # Obstacle avoidance penalty and jumping reward
    obstacle_positions = [2.0, 10.0, 13.0]
    obstacle_penalty = 0.0
    jumping_reward = 0.0

    for obs_x in obstacle_positions:
        distance_to_obstacle = abs(x_position_after - obs_x)
        if distance_to_obstacle < 1.0:
            obstacle_penalty -= 1.0 / distance_to_obstacle
            if z_position_after > 0.5:
                jumping_reward += 1.0

    reward = forward_reward - ctrl_cost + angle_penalty - obstacle_penalty + jumping_reward
```

This custom reward function combines multiple elements: it provides a base reward for forward movement, penalizes the agent for not maintaining an upright posture, adds a penalty for being close to obstacles, and gives a reward for jumping over obstacles.

2.5 Distance comparison, Conclusion

With given default settings, as in the default output video, the cheetah struggles to move over obstacles and is even heading towards the opposite direction. However, with my custom functions and parameter tuning, the cheeta moves approximately **9 square blocks/meters**, and also is capable of jumping over the obstacles. The experiments revealed that modifying the discount factor, exploration noise, observation space, and the reward function significantly improves the performance of the Half Cheetah agent in an environment with obstacles. The custom reward function played a crucial role in teaching the agent to avoid obstacles effectively. Moreover, including the distance and height of the nearest obstacle in the observation space enabled the agent to make better-informed decisions regarding when and how to jump.

References

[1] E. T. T. E. Y. Tassa, “Mujoco: A physics engine for model-based control,” 2012. Accessed: 2023-06-19.

[2] R. A. R. M. K. Ciosek, “Discount factor as a regularizer in reinforcement learning,” 2020. Accessed: 2023-06-20.

[3] M. Fortunato, “Noisy networks for exploration,” 2020. Accessed: 2023-06-20.